

A Formalisation of Constraint-Aware Model Transformations

Adrian Rutle¹, Alessandro Rossini², Yngve Lamo¹, and Uwe Wolter²

¹ Bergen University College, P.O. Box 7030, 5020 Bergen, Norway
{aru,yla}@hib.no

² University of Bergen, P.O. Box 7803, 5020 Bergen, Norway
{rossini,wolter}@ii.uib.no

Abstract. This paper introduces a formal approach to the definition of constraint-aware model transformations. The proposed approach is based on the Diagram Predicate Framework and extends graph transformations with the ability to handle constraints in the definition and execution of model transformations. In particular, it uses non-deleting rules that are typed over the metamodel of a joined modelling language which is constructed from the source and target languages. Furthermore, the application of transformation rules is formalised as a pushout construction that creates a model which is typed over the metamodel of the joined modelling language. Finally, the target model is obtained from the created model by a pullback construction.

1 Introduction and Motivation

Models are first-class entities of the software development process in Model-Driven Engineering (MDE) and undergo a complex evolution during their life-cycles. In this regard, model transformation is one of the key techniques which is used to automate several model-based activities such as code generation, refactoring, optimisation, language translation etc. [23].

A general definition of model transformation given in [11] and further generalised in [15] is as follows. A *transformation* is the automatic generation of target models from source models, according to a transformation definition. A *transformation definition* is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A *transformation rule* is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language.

Several classifications of model transformations are given in [4,15]. A first classification is based on whether the transformation is used to transform models specified by one modelling language, called *homogeneous* transformation, or models specified by different modelling languages, called *heterogeneous* transformation. The former class of transformations is suitable for model refactoring and optimisation [1], while the latter is suitable for language translation. A second classification is based on whether the target model is created from scratch, called *out-place*, or the source model is modified in order to obtain the target model, called *in-place*. A third classification is based on the

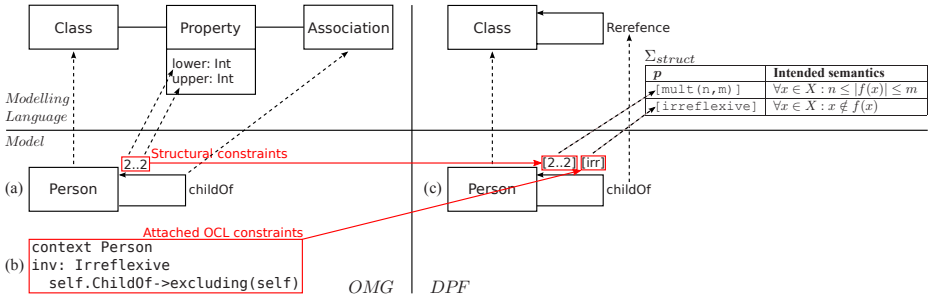


Fig. 1. Constraints in MOF-based modelling languages and DPF: (a) structural constraints in UML (b) attached OCL constraints (c) integration of constraints in DPF

underlying technique which is used to carry out the transformations, e.g. logic programming, functional programming, graph transformation, etc. The approach introduced in this paper is classified under heterogeneous, out-place and graph-transformation based transformation.

In the context of MDE, models are typically specified by means of modelling languages such as the Unified Modeling Language (UML) [17]. Each of these modelling languages has a corresponding metamodel – a model that defines the abstract syntax of models which can be specified by the modelling language. These metamodels, in turn, are specified by means of a metamodeling language called the Meta-Object Facility (MOF) [16]. MOF-based modelling languages allow the specification of simple constraints such as multiplicity and uniqueness constraints, hereafter called *structural constraints*. These constraints are usually specified by properties of classes in the metamodel of the modelling language. For instance, the requirement “a person is the child of exactly two parents” in a UML model can be forced by a multiplicity constraint which uses the properties *lower* and *upper* of the class *Property* of the UML metamodel (see Fig. 1a). Instances of the UML model should satisfy this multiplicity constraint. However, these structural constraints may not be sufficient to meet complex requirement’s specifications. Hence, textual constraint languages such as the Object Constraint Language (OCL) are usually used to define complex constraints, hereafter called *attached OCL constraints*. For instance, the requirement “a person can not be a child of her/himself” in a UML model can only be forced by an OCL expression (see Fig. 1b).

While existing model transformation techniques always take into account structural constraints, they often ignore the attached OCL constraints [14,18]. This is because model transformation rules are defined over metamodel elements while attached OCL constraints are defined in a different technical space. This problem is closely related to the fact that the *conformance* relation between models and metamodels is not formally defined for MOF-based modelling languages [5,19], especially when OCL constraints are involved [2].

In this paper, a solution to this challenge is proposed. The solution is based on the Diagram Predicate Framework (DPF) [21,20,22] and reuses the diagrammatic formalisation of MOF-based modelling languages described in [21]. DPF provides a formal diagrammatic approach to (meta)modelling and model transformation based on

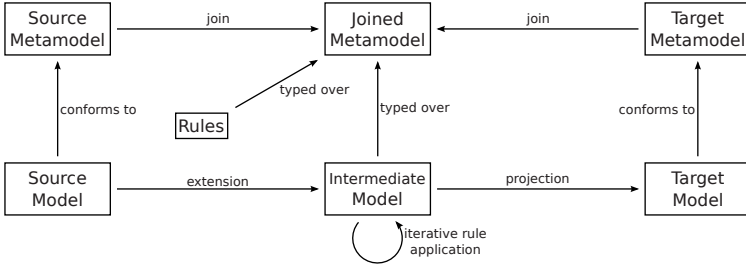


Fig. 2. Overview of the DPF-based approach to model transformation

category theory [9]. The proposed approach to model transformation provides an extension to the formal framework of graph transformations [8] in the sense that it can be used to transform models as well as attached OCL constraints. This is done by integrating structural constraints and attached OCL constraints in modelling formalisms which represent the formalisation of MOF-based modelling languages (see Fig. 1c for an intuition).

The DPF-based approach to model transformation introduces the concept of constraint-aware model transformation; i.e. a model transformation technique which supports specifying constraints in input and output patterns and using these constraints to control (i) which structure to create in the target model and (ii) which constraints to impose on the created structure. The first step in this approach consists of creating a joined modelling language which consists of a combination of the source and the target modelling languages; i.e. a joined metamodel is created (see Fig. 2). The second step consists of defining constraint-aware model transformation rules which are typed over the joined metamodel. The model transformation is applied in a final step as follows: extending a source model which conforms to the source metamodel to an intermediate model which is typed over the joined metamodel, iterative application of the transformation rules, and projection of a target model which conforms to the target metamodel.

A running example is used to illustrate our approach. It presents a transformation of an object-oriented structural model to a relational data model. In this example, the syntax used for the definition of the transformation rules is the same as the syntax used to specify the (meta)models.

The remainder of the paper is organised as follows. Section 2 reviews our diagrammatic formalisation of MOF-based modelling languages. Section 3 presents the details of our model transformation approach. In Section 4, a comparison of our approach with other graph transformation-based approaches to model transformation is given. Finally, in Section 5 some concluding remarks and ideas for future work are presented.

2 Diagram Predicate Framework

In DPF, models are formalised as *diagrammatic specifications*. A diagrammatic specification $S = (G^S, C^S)$ consists of an underlying graph G^S decorated by a set of constraints C^S . The graph represents the structure of the model, and predicates from a predefined *diagrammatic predicate signature* are used to impose constraints on the

model. In this paper, we use the terms “model” and “specification” interchangeably. The formal definitions are as follows:

Definition 1 (Signature). A (diagrammatic predicate) signature $\Sigma := (\Pi, \alpha)$ consists of a collection of predicate symbols Π with a mapping α that assigns a graph to each predicate symbol $p \in \Pi$. $\alpha(p)$ is called the arity of the predicate symbol p .

Definition 2 (Constraint). Given a signature $\Sigma = (\Pi, \alpha)$, a constraint (p, δ) on a graph G is given by a predicate symbol p and a graph homomorphism $\delta : \alpha(p) \rightarrow G$.

Definition 3 (Σ -specification). Given a signature $\Sigma = (\Pi, \alpha)$, a (diagrammatic) Σ -specification $S := (G^S, C^S)$ is given by a graph G^S and a set C^S of constraints (p, δ) on G^S with $p \in \Pi$.

Example 1. Let us consider an information system for the management of employees and projects. At any state of the system the following requirements should be satisfied.

1. An employee must work for at least one department.
2. A department may have none or many employees.
3. A project may involve none or many employees.
4. A project must be controlled by at least one department.
5. An employee involved in a project must work in the controlling department.

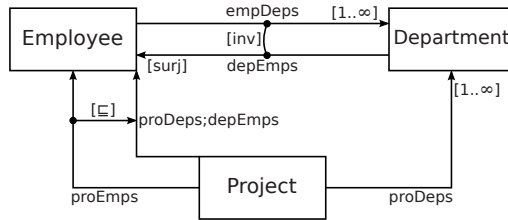


Fig. 3. A Σ_{struct} -specification $S = (G^S, C^S)$

Fig. 3 shows a Σ_{struct} -specification $S = (G^S, C^S)$ which specifies a structural model compliant with the requirements above. Table 1 shows the signature Σ_{struct} . Here we present how two of the above mentioned requirements are specified in S using the predicates from Σ_{struct} . In particular we present a requirement which can be specified by means of structural constraints in UML syntax; as well as a requirement which demands the usage of attached OCL constraints (see [21] for a comparison of the UML/OCL- and DPF-based models of a similar system). The requirement “an employee must work for at least one department” is forced in S by the predicate $[\text{mult}(1, \infty)]$ on the arrow `empDeps`. Furthermore, the requirement “an employee involved in a project must work in the controlling department” is forced in S by using the predicates $[\text{composition}]$ and $[\text{subset}]$ on the arrows `proDeps; depEmps` and `proEmps`.

Table 1. The signature Σ_{struct}

p	$\alpha_{struct}(p)$	Proposed visual.	Intended semantics
[mult(n,m)]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[f]{[n,m]} \boxed{Y}$	$\forall x \in X : n \leq f(x) \leq m$
[irreflexive]	$1 \xrightarrow{f} 1$	$\boxed{X} \xrightarrow{f} \boxed{X}$	$\forall x \in X : x \notin f(x)$
[injective]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[f]{[inj]} \boxed{Y}$	$\forall x, x' \in X : f(x) = f(x')$ implies $x = x'$
[surjective]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[f]{[surj]} \boxed{Y}$	$f(X) = Y$
[jointly-injective]	$1 \xrightarrow{f} 2$ $g \downarrow$ 3	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $g \downarrow$ \boxed{Z} [ji]	$\forall x, x' \in X : f(x) = f(x')$ and $g(x) = g(x')$ implies $x = x'$
[inverse]	$1 \xrightarrow{f} 2$ $g \xrightarrow{f} 1$	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $g \xrightarrow{f} \boxed{X}$ [inv]	$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$
[composition]	$1 \xrightarrow{f} 2$ $h \searrow$ $g \downarrow$ 3	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $g \downarrow$ \boxed{Z} $f;g \searrow$	$\forall x \in X : f;g(x) = \bigcup \{g(y) \mid y \in f(x)\}$
[subset]	$1 \xrightarrow{f} 2$ $g \xrightarrow{f} 1$	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $g \xrightarrow{f} \boxed{X}$ [sq]	$\forall x \in X : f(x) \subseteq g(x)$

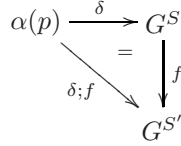
Note that, any OCL-constraint that can be seen as a “sort-wise” property; i.e. properties of sets, functions, or diagrams of sets and functions as a whole, can be specified in DPF. This is because DPF is based on category theory which is centred around sort-wise properties. A precise characterisation of non-sort-wise OCL-constraints is an open issue which is part of our current research.

In DPF, we distinguish between two types of conformance relations: *typed over* and *conforms to*. A model is typed over a metamodel if its underlying graph is typed over the underlying graph of the metamodel; i.e. each model element is assigned a type in the metamodel by a *typing morphism*. In contrast, a model is said to conform to a metamodel if it is typed over the metamodel and, in addition, it satisfies all the constraints of the metamodel [21].

The definition of typed Σ -specification depends on the definition of Σ -specification morphisms. These definitions will also be used in Section 3 in our approach to model transformation. A specification morphism between two Σ -specifications is a graph homomorphism which preserves constraints. In contrast, a typed specification morphism

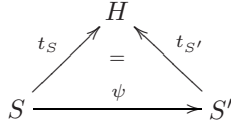
between two typed Σ -specifications is a specification morphism which respects the typing morphisms. The formal definitions are as follows:

Definition 4 (Σ -specification Morphism). A Σ -specification morphism $f : S \rightarrow S'$ between two Σ -specifications $S = (G^S, C^S)$ and $S' = (G^{S'}, C^{S'})$ is a graph homomorphism $f : G^S \rightarrow G^{S'}$ preserving constraints, i.e. $(p, \delta) \in C^S$ implies $(p, \delta; f) \in C^{S'}$ for all constraints $(p, \delta) \in C^S$.



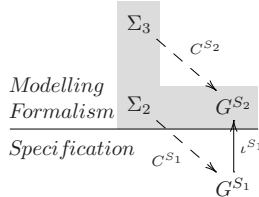
Definition 5 (Typed Σ -specification). An H -typed Σ -specification S , i.e. a Σ -specification S together with a graph homomorphism $t^S : G^S \rightarrow H$. t^S is called a typing morphism.

Definition 6 (Typed Σ -specification Morphism). A typed Σ -specification morphism between two H -typed Σ -specifications S and S' is a Σ -specification morphism $\psi : S \rightarrow S'$ such that $\psi; t^{S'} = t^S$.



In DPF, each modelling language is formalised as a modelling formalism which is a triple $F = (\Sigma_2, S_2, \Sigma_3)$ [21]. The concepts of the modelling language are located in the Σ_3 -specification S_2 ; and the constraining constructs which are available for the users of the modelling language are located in the signature Σ_2 .

Definition 7 (Modelling Formalism). A modelling formalism $F = (\Sigma_2, S_2, \Sigma_3)$ is given by signatures $\Sigma_2 = (\Pi_2, \alpha_2)$ and $\Sigma_3 = (\Pi_3, \alpha_3)$, and a Σ_3 -specification $S_2 = (G^{S_2}, C^{S_2})$. S_2 is called the metamodel of F . An F -specification is a Σ_2 -specification $S_1 = (G^{S_1}, C^{S_1})$ which conforms to S_2 .



Predicates from the signature Σ_3 are used to add constraints to the metamodel S_2 . This corresponds to metamodel definition. These constraints should be satisfied by F -specifications. Moreover, predicates from the signature Σ_2 are used to add constraints to F -specifications. This corresponds to model definition. These constraints should be satisfied by instances of the F -specifications, in the same way as the F -specifications should satisfy the metamodel constraints.

For a given modelling formalism, the semantics of nodes and arrows have to be chosen in a way which is appropriate for the corresponding modelling environment. For structural models in object-oriented development, it is appropriate to interpret nodes as sets and arrows $X \xrightarrow{f} Y$ as multi-valued functions $f : X \rightarrow \wp(Y)$. The powerset $\wp(Y)$ of Y is the set of all subsets of Y , i.e. $\wp(Y) = \{K \mid K \subseteq Y\}$. On the other hand, for relational data models it is appropriate to interpret nodes as sets and arrows as single-valued functions.

3 Model Transformation

This section describes the DPF-based approach to constraint-aware model transformation. The first step consists of creating a joined modelling formalism which enables the specification of both source and target models. One way to achieve this is to construct the disjoint union of the components of the source and target modelling formalisms.

Roughly speaking, given the source $F^S = (\Sigma_2^S, S_2, \Sigma_3^S)$ and the target $F^T = (\Sigma_2^T, T_2, \Sigma_3^T)$ modelling formalisms (see Fig. 4a and Fig. 4c, respectively), a joined modelling formalism $F^J = (\Sigma_2^J, J_2, \Sigma_3^J)$ will be created (see Fig. 4b). In more detail, the source and target metamodels are joined together to $J_2 := S_2 \uplus K_2 \uplus T_2$, and the source and target signatures are joined together to $\Sigma_2^J := \Sigma_2^S \uplus \Sigma_2^T$ and $\Sigma_3^J := \Sigma_3^S \uplus \Sigma_3^K \uplus \Sigma_3^T$, where \uplus denotes the disjoint union operation (see Example 2). In J_2 , the component K_2 represents the correspondence between S_2 and T_2 . In most cases, the elements in K_2 will be arrows connecting nodes in S_2 and T_2 . However, in some cases it may be convenient to have also auxiliary nodes in K_2 and arrows connecting these nodes with elements in S_2 and/or T_2 . In Σ_3^J , the component Σ_3^K contains additional predicates which are used to constrain elements of J_2 . The definitions of K_2 and Σ_3^K should be done manually by transformation designers.

Although the transformation designer is free to relate any elements of the source and target metamodels, there is a *projection condition* which should be satisfied by the joined metamodel J_2 . The condition is that for any F^J -specification J_1 , it should be possible to construct an F^S -specification S_1 and an F^T -specification T_1 by pullback

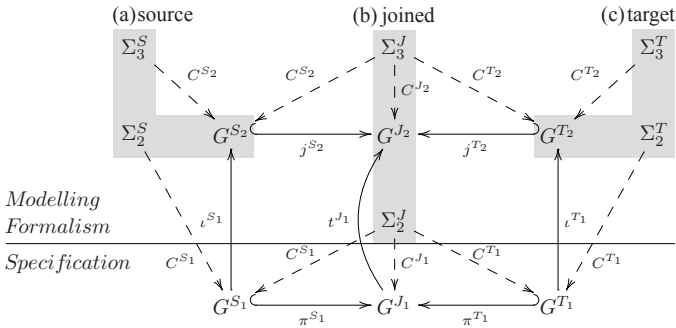


Fig. 4. Source, target and joined modelling formalisms

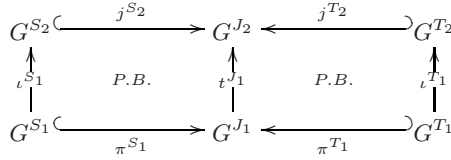


Fig. 5. Projection condition for the joined metamodel J_2

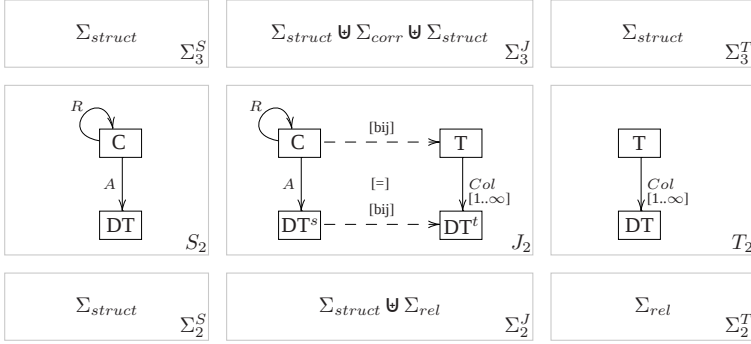


Fig. 6. Joining the modelling formalisms of structural models and relational data models

constructions as depicted in Fig. 5 (see [6] for a description and motivation for using pullbacks in model transformation). In order to satisfy the projection condition, J_2 must be constrained by predicates from the signature Σ_3^J .

In the next example, we present a joined modelling formalism and show the constraints of J_2 which are necessary to satisfy the projection condition.

Example 2. Fig. 6 shows a source modelling formalism $F^S = (\Sigma_2^S, S_2, \Sigma_3^S)$ for specifying structural models; a target modelling formalism $F^T = (\Sigma_2^T, T_2, \Sigma_3^T)$ for specifying relational data models; and their joined modelling formalism $F^J = (\Sigma_2^J, J_2, \Sigma_3^J)$. In particular, $\Sigma_3^S, \Sigma_3^T, \Sigma_2^S := \Sigma_{struct}$ (see the signature Σ_{struct} for structural models in Table 1); $\Sigma_2^T := \Sigma_{rel}$ (see part of the signature Σ_{rel} for relational data models in Table 2); $\Sigma_2^J := \Sigma_{struct} \uplus \Sigma_{rel}$; and $\Sigma_3^J := \Sigma_{struct} \uplus \Sigma_{corr} \uplus \Sigma_{struct}$ (see part of the signature Σ_{corr} for defining correspondence constraints in Table 3). In the source metamodel S_2 , the arrows R and A stand for Reference and Attribute, respectively; and the nodes C and DT stand for Class and Data Type, respectively. In the target metamodel T_2 , the arrow Col stand for Column; and the nodes T and DT stand for Table and Data Type, respectively. Note that the node DT is renamed to DT^s and DT^t in the joined metamodel J_2 by the disjoint union operation. The projection condition is satisfied by constraining the arrows in J_2 by the predicates [bijective] and [commutative]. This means that for each class, a corresponding table should be created. In addition, for each attribute belonging to a class, a corresponding column belonging to the corresponding table should be created.

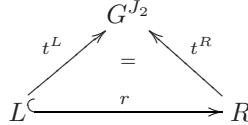
Table 2. A part of the signature Σ_{rel}

p	$\alpha_{rel}(p)$	Proposed visual.	Intended semantics
[total]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow{f} \boxed{Y}$	$\forall x \in X : f(x) = 1$
[primary-key]	$1 \xrightarrow{f} 2$	$\boxed{X} \xrightarrow[\text{[pk]}]{f} \boxed{Y}$	f is [total] and [injective]
[foreign-key]	$1 \xrightarrow{f} 2$ $\uparrow g$ 3	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $\uparrow \text{[fk]}$ $\boxed{Z} \xrightarrow{g} \boxed{Y}$	$f(X) \subseteq g(Y)$
[image-equal]	$1 \xrightarrow{f} 2$ $\uparrow g$ 3	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $\uparrow \text{[ie]}$ $\boxed{Z} \xrightarrow{g} \boxed{Y}$	$f(X) = g(Z)$
[join]	$1 \xrightarrow{f} 2$ $\uparrow g'$ 3 $\xrightarrow{f'}$ 4 $\uparrow g$	$\boxed{X} \xrightarrow{f} \boxed{Y}$ $\uparrow g'$ $\boxed{XZ} \xrightarrow{f'} \boxed{Z}$ $\uparrow g$ [join]	$\forall x \in X, \forall z \in Z : (x, z) \in XZ$ iff $f(x) = g(z)$

3.1 Constraint-Aware Transformation Rules

The second step in our approach is the definition of constraint-aware transformation rules. In each transformation rule, the input pattern is included in the output pattern. The input and output patterns are Σ_2^J -specifications which are typed over G^{J_2} .

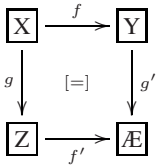
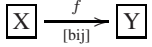
Definition 8 (Transformation Rules). *Given a joined modelling formalism $F^J = (\Sigma_2^J, J_2, \Sigma_3^J)$, a transformation rule $r : L \hookrightarrow R$ is a G^{J_2} -typed Σ_2^J -specification morphism between the input and output patterns L and R , with r being an inclusion.*



In the following example, we illustrate our approach to the definition of transformation rules. Note that the output patterns of the transformation rules are not only dependent on the structure of the input patterns, but also on the constraints.

Example 3. Building on Example 2, Table 4 outlines some of the transformation rules which are used for the transformation of structural models to relational data models. These rules are typed over the joined metamodel J_2 shown in Fig. 6. Note that $(1:C)$ is a “user-friendly” notation for the typing assignment $(t : 1 \mapsto C)$. In rule r_1 , each

Table 3. A part of the signature Σ_{corr}

p	$\alpha_{corr}(p)$	Proposed visual.	Intended semantics
[commutative]	$ \begin{array}{ccc} 1 & \xrightarrow{f} & 2 \\ g \downarrow & & \downarrow g' \\ 3 & \xrightarrow{f'} & 4 \end{array} $		$\forall x \in X : g'(f(x)) = f'(g(x))$
[bijective]	$ 1 \xrightarrow{f} 2 $		f is [mult(1,1)], [injective] and [surjective]

class is transformed to a corresponding table. In rule r_2 , for each attribute a column is created. The rules r_3 and r_4 are used to transform bidirectional references (or a pair of inverse functions) between two classes to foreign keys between two tables. Notice that the only difference between the input patterns of the rules r_3 and r_4 is the constraint forced by the predicate [mult(0, 1)] on the arrow 2:R. This constraint affects the way in which a match of the input pattern is transformed to a match of the output pattern. More precisely, since in r_3 each 2:C is related to at most *one* 1:C, a foreign key column 3:Col will be created which will refer to 1:Col. However, in r_4 each 2:C may be related to many 1:C and vice versa. Therefore, a link table 3:T is created with two foreign key columns 3:Col and 4:Col. The contents of this link table may be seen as tuples (1:DT^t, 2:DT^t).

3.2 Application of Model Transformation

The last step in our approach is the application of model transformation. In this step categorical constructions [7,9] such as pushout and pullback are exploited. The application of a model transformation consists of iterative application of transformation rules.

Definition 9 (Application of Transformation Rules). Given a source model J_1 , an application $\langle r, m \rangle$ of a transformation rule $r : L \hookrightarrow R$ via a match $m : L \rightarrow J_1$, where m is a G^{J_2} -typed Σ_2^J -specification morphism, is given by the pushout

$$\begin{array}{ccc}
 L \hookrightarrow R & \xrightarrow{r} & R \\
 m \downarrow & \text{P.O.} & \downarrow m^* \\
 J_1 \hookrightarrow J_1' & \xrightarrow{\langle r, m \rangle} & J_1'
 \end{array}$$

In the following, the procedure for transforming a source model S_1 to a target model T_1 is outlined.

1. *Extension of the source model.* The source F^S -specification S_1 is extended to an intermediate G^{J_2} -typed Σ_2^J -specification J_1 . This transformation is given by the composition $\iota^{S_1}; j^{S_2}$ (see Fig. 4) which leads to $J_1 = S_1 \uplus K_1 \uplus T_1$ with both K_1 and T_1 being empty specifications.

2. *Iterative application of the transformation rules.* Upon the application of a rule $r : L \leftrightarrow R$, for a match of the input pattern L in $J_1 = S_1 \uplus K_1 \uplus T_1$, the K_1 and T_1 parts will be extended by an appropriate copy of the new elements in R , i.e., by those elements in R that are not already in L .
3. *Obtaining the target model.* The iterative application of transformation rules may stop once an F^J -specification J_1 is constructed. The projection condition ensures then that the pullback of the span $G^{J_1} \rightarrow G^{J_2} \leftarrow G^{T_2}$ (see Fig. 5) constructs an F^T -specification T_1 which may be considered the target model.

In Table 4, we defined the transformation rules which were needed to transform structural models (with only structural constraints) to relational data models. What remains to show now is the ability to define transformation rules which enable transformation of more complex constraints such as the requirement 5 in Example 1.

Example 4. Building on Example 3, Fig. 7 shows the relational data model which is created by applying the rules in Table 4 and 5 to the Σ_{struct} -specification S in Fig. 3. Recall that arrows in Σ_{rel} -specifications are interpreted as single-valued functions. Hence, we do not need to add constraints to force single-valued functions.

Table 4. Rules for the transformation of structural models to relational data models

L	R
Rule r_1 . Class to table $\boxed{1:C}$	$\boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t}$
Rule r_2 . Attribute to column $\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \downarrow 1:A \\ \boxed{1:DT^s} \end{array}$	$\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \downarrow 1:A \quad \downarrow 2:Col \\ \boxed{1:DT^s} \dashrightarrow \boxed{1:DT^t} \end{array}$
Rule r_3 . Many-to-one reference to foreign key $\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \uparrow [0..1] \\ \begin{array}{c} 1:R \xrightarrow{[inv]} 2:R \\ \downarrow \\ \boxed{2:C} \dashrightarrow \boxed{2:T} \xrightarrow[2:Col]{[pk]} \boxed{Int:DT^t} \end{array} \end{array}$	$\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \uparrow [0..1] \\ \begin{array}{c} 1:R \xrightarrow{[inv]} 2:R \\ \downarrow \\ \boxed{2:C} \dashrightarrow \boxed{1:T} \xrightarrow[2:Col]{[pk]} \boxed{Int:DT^t} \end{array} \end{array}$ <p style="text-align: center;">$\swarrow [fk]$</p>
Rule r_4 . Many-to-many reference to link table and foreign keys $\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \uparrow [inv] \\ \begin{array}{c} 1:R \xrightarrow{[inv]} 2:R \\ \downarrow \\ \boxed{2:C} \dashrightarrow \boxed{2:T} \xrightarrow[2:Col]{[pk]} \boxed{Int:DT^t} \end{array} \end{array}$	$\begin{array}{c} \boxed{1:C} \dashrightarrow \boxed{1:T} \xrightarrow[1:Col]{[pk]} \boxed{Int:DT^t} \\ \uparrow [inv] \\ \begin{array}{c} 1:R \xrightarrow{[inv]} 2:R \\ \downarrow \\ \boxed{2:C} \dashrightarrow \boxed{2:T} \xrightarrow[2:Col]{[pk]} \boxed{Int:DT^t} \end{array} \end{array}$ <p style="text-align: center;">$\uparrow [fk]$ $\uparrow [fk]$ $\uparrow [fk]$</p> <p style="text-align: center;">$\begin{array}{c} \boxed{3:T} \\ \downarrow 3:Col \\ \downarrow 4:Col \end{array}$</p>

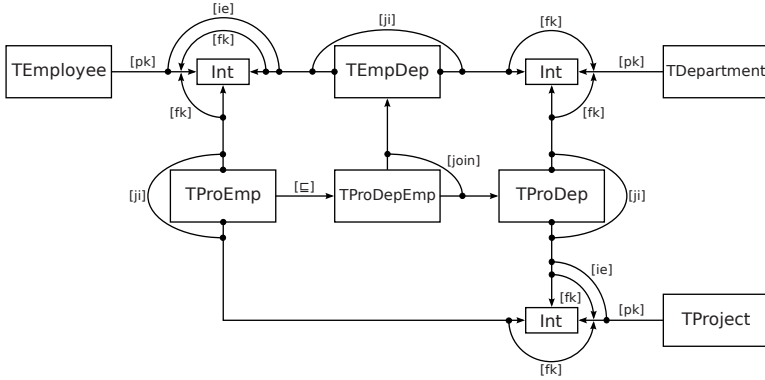


Fig. 7. The target Σ_{rel} -specification resulting from the transformation of the Σ_{struct} -specification presented in Fig. 3

However, we use the predicate $[total]$ from Σ_{rel} to add constraints which force total functions whenever necessary, for example columns for which a value is required. The rule r_5 in Table 5 is applied to the arrows $empDeps$ and $depEmps$, and the predicates $[inverse]$ and $[surjective]$ are transformed to $[foreign-key]$, $[image-equal]$, $[total]$ and $[jointly-injective]$ on the arrows connecting the nodes **TEmployee**, **TEmpDep** and **TDepartment** to **Int** in Fig. 7. The predicate $[image-equal]$ is used to force that for any row in the table **TEmployee** there is a corresponding row in the table **TEmpDep**. Moreover, the rule r_6 is applied to the arrows $proEmps$ and $proDeps$; $depEmps$, and the predicates $[composition]$ and $[subset]$ are transformed to $[join]$ and $[subset]$ on the arrows connected to the node **TProDepEmp**.

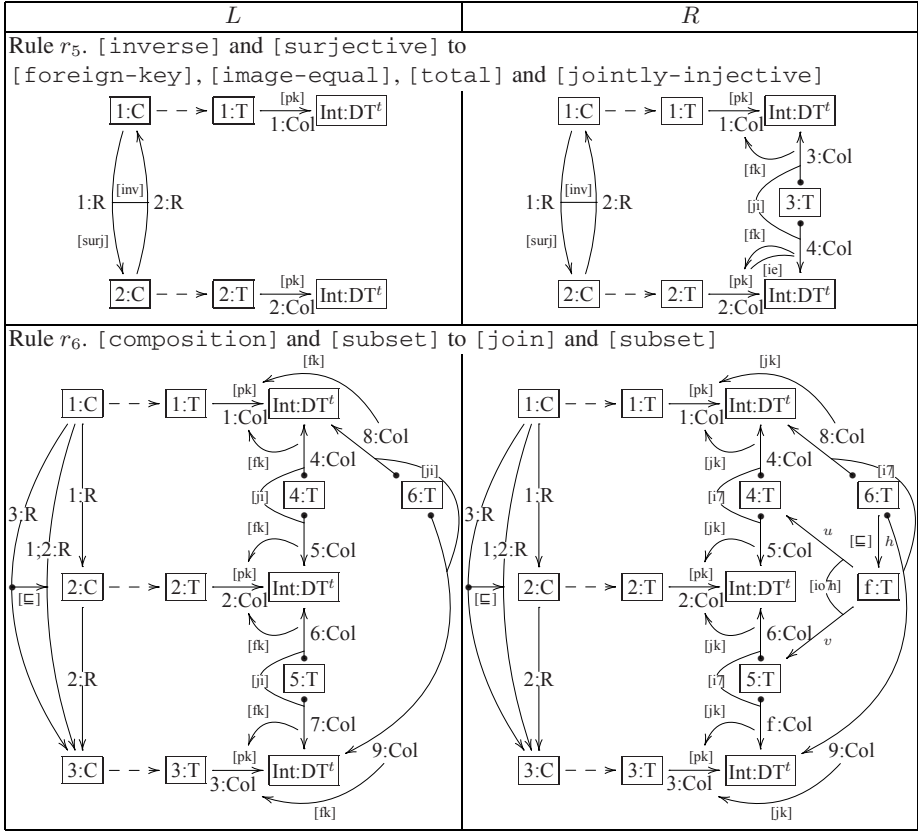
4 Related Work

Several approaches to transform graph-based structures have been developed in the field of graph grammars and graph transformations. In this section, a short comparison of our approach to some graph transformation-based approaches is outlined.

Graph Transformation Systems (GTS) [8] are widely used as the formal foundation for model transformation approaches. In GTS, models are represented by typed attributed graphs, and the conformance relation between models and metamodels is given by a typing morphism. In DPF, models and metamodels are additionally equipped with a set of constraints, and a model conforms to a metamodel if there exists a typing morphism that satisfies the constraints in the metamodel. In the context of model transformation, we extend GTS by adding support for transformation of the constraints which come additional to the graph structure of the models.

Triple Graph Grammar (TGG) [12,7] is a GTS-based approach which is suitable for language translation. A triple graph consists of a source and a target graph that are related via a correspondence graph and two graph homomorphisms from the correspondence graph to the source and target graphs. In this way, the source and target graphs

Table 5. Rules for the transformation of attached constraints



are joined into a single structure thus providing a basis for consistent co-evolution of the graphs [7]. The use of correspondence graphs allows to relate a node (arrow) in the source graph with a node (arrow) in the target graph and to constrain these simple relations by means of OCL. Similar to TGG, in DPF a joined metamodel is used to describe relations between the source and target metamodels. The difference is however that we can define and constrain, in a diagrammatic way, arbitrary complex relations between source and target metamodel elements, e.g. the commutativity constraint in Fig. 6.

The Visual Modeling and Transformation System (VMTS) [13] is an n-layer meta-modelling environment which supports editing models according to their metamodels and allows specifying OCL constraints. VMTS provides a graph transformation-based approach to model transformations in which models are formalised as directed, labelled graphs. Moreover, OCL constraints are used to control the execution of transformations. The input and output patterns of transformation rules use metamodel elements; meaning that an instantiation of the input pattern must be found in the source graph instead of an isomorphic subgraph of the pattern. These patterns are guarded by pre- and post-conditions. Before the execution of each transformation rule, the pre-conditions are

checked and used to narrow down the set of matches. After execution of each rule, the post-conditions are checked against the output of the rule. In this way, if a rule executes successfully it can be asserted that the transformation has resulted in the expected output. The DPF-based approach is different in that constraints are not used only for controlling the matches, but they can also be transformed from the source models to target models. In DPF terms, this means that VTMS can be seen as transforming the set of constraints C^{S_2} while ignoring C^{S_1} .

An approach to the analysis of graph transformation rules based on an intermediate OCL representation is presented in [3]. The semantics of rules together with their properties (such as rule applicability, conflict or independence) are transformed into OCL expressions. While these OCL expressions are combined with structural- and attached OCL constraints during the analysis process, the attached OCL constraints are not shown to be transformed. In the DPF-based approach attached OCL constraints and structural constraints are integrated in modelling formalisms, facilitating a uniform transformation of these constraints.

The approach proposed in [10] employs transformation rules to preserve the semantics of UML/OCL class diagrams when using the refactoring rule `moveAttribute`. The DPF-based approach is more generic in the sense that it can be used for the definition of transformation rules between constrained models which are specified by different modelling languages.

5 Conclusion and Future Work

This paper proposes a formal approach to the definition of constraint-aware model transformation which is applied to language translation. This is possible due to the diagrammatic formalisation of MOF-based modelling languages in which attached OCL constraints are integrated in modelling formalisms.

In this approach, the process of model transformation is organised into three steps. Firstly, the source and target modelling languages are joined together; i.e. a joined metamodel is created. Secondly, the transformation rules are declared as input and output patterns which are typed over the joined metamodel. The input and output patterns of the transformation rules are diagrammatic specifications; and the morphisms between input and output patterns as well as their matches are formalised as constraint- and type preserving specification morphisms. Hence, constraints can be added to the input patterns, and these constraints can be used to control (i) which structure to create in the target model and (ii) which constraints to add to the created structure. Thirdly, the model transformation is applied as follows. The source model is extended to an intermediate model which is typed over the joined metamodel. Next, the transformation rules are iteratively applied to the intermediate model. Finally, the target model is obtained by projection. The approach exploits existing machinery from category theory to formalise constraint-aware model transformations. More precisely, pushout construction is used for the application of transformation rules, and pullback construction is used for the projection of target models.

In a future study, we will analyse scheduling and controlling application of constraint-aware transformation rules building upon our previous work described in [22]. Furthermore, analysing to what extent our approach is suitable for bidirectional transformations is part of our current research.

References

1. Biermann, E., Ehrig, K., Köhler, C., Kuhns, G., Taentzer, G., Weiss, E.: EMF Model Refactoring based on Graph Transformation Concepts. ECEASST 3 (2006)
2. Boronat, A., Meseguer, J.: Algebraic Semantics of OCL-Constrained Metamodel Specifications. In: Oriol, M., Meyer, B. (eds.) TOOLS Europe 2009: 47th International Conference on Objects, Components, Models and Patterns. LNBP, vol. 33, pp. 96–115. Springer, Heidelberg (2009)
3. Cabot, J., Clarisó, R., Guerra, E., de Lara, J.: Analysing Graph Transformation Rules through OCL. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) ICMT 2008. LNCS, vol. 5063, pp. 229–244. Springer, Heidelberg (2008)
4. Czarnecki, K., Helsen, S.: Classification of Model Transformation Approaches. In: OOPSLA 2003: 2nd Workshop on Generative Techniques in the Context of MDA (2003)
5. Diskin, Z.: Mathematics of UML: Making the Odysseys of UML less dramatic. In: Practical foundations of business system specifications, pp. 145–178. Kluwer Academic Publishers, Dordrecht (2003)
6. Diskin, Z., Dingel, J.: A metamodel Independent Framework for Model Transformation: Towards Generic Model Management Patterns in Reverse Engineering. Technical Report 1/2006, ATEM 2006, Johannes Gutenberg Universität Mainz, Germany (October 2006)
7. Ehrig, H., Ehrig, K., Ermel, C., Hermann, F., Taentzer, G.: Information Preserving Bidirectional Model Transformations. In: Dwyer, M.B., Lopes, A. (eds.) FASE 2007. LNCS, vol. 4422, pp. 72–86. Springer, Heidelberg (2007)
8. Ehrig, H., Ehrig, K., Prange, U., Taentzer, G.: Fundamentals of Algebraic Graph Transformation. Springer, Heidelberg (2006)
9. Fiadeiro, J.L.: Categories for Software Engineering. Springer, Heidelberg (2004)
10. Fondement, F., Baar, T.: Making Metamodels Aware of Concrete Syntax. In: Hartman, A., Kreische, D. (eds.) ECMDA-FA 2005. LNCS, vol. 3748, pp. 190–204. Springer, Heidelberg (2005)
11. Kleppe, A.G., Warmer, J., Bast, W.: MDA Explained: The Model Driven Architecture: Practice and Promise. Addison-Wesley Longman Publishing Co., Inc., Boston (2003)
12. Königs, A., Schürr, A.: Tool Integration with Triple Graph Grammars – A Survey. ENTCS 148(1), 113–150 (2006)
13. Lengyel, L., Levendovszky, T., Charaf, H.: Constraint Validation Support in Visual Model Transformation Systems. Acta Cybernetica 17(2), 339–357 (2005)
14. Marković, S., Baar, T.: Refactoring OCL annotated UML class diagrams. Software and System Modeling 7(1), 25–47 (2008)
15. Mens, T., Gorp, P.V.: A Taxonomy of Model Transformation. ENTCS 152, 125–142 (2006)
16. Object Management Group: Meta-Object Facility Specification (January 2006), <http://www.omg.org/cgi-bin/doc?formal/2006-01-01>
17. Object Management Group: Unified Modeling Language Specification (February 2009), <http://www.omg.org/cgi-bin/doc?formal/2009-02-04>
18. Petter, A., Behring, A., Mühlhäuser, M.: Solving Constraints in Model Transformations. In: Paige, R.F. (ed.) ICMT 2009. LNCS, vol. 5563, pp. 132–147. Springer, Heidelberg (2009)

19. Poernomo, I.H.: A Type Theoretic Framework for Formal Metamodelling. In: Reussner, R., Stafford, J.A., Szyperski, C. (eds.) *Architecting Systems with Trustworthy Components*. LNCS, vol. 3938, pp. 262–298. Springer, Heidelberg (2006)
20. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A Category-Theoretical Approach to the Formalisation of Version Control in MDE. In: Chechik, M., Wirsing, M. (eds.) *FASE 2009*. LNCS, vol. 5503, pp. 64–78. Springer, Heidelberg (2009)
21. Rutle, A., Rossini, A., Lamo, Y., Wolter, U.: A Diagrammatic Formalisation of MOF-Based Modelling Languages. In: Brakhage, H. (ed.) *TOOLS Europe 2009: 47th International Conference on Objects, Components, Models and Patterns*. LNBIP, vol. 33, pp. 37–56. Springer, Heidelberg (2009)
22. Rutle, A., Wolter, U., Lamo, Y.: A Diagrammatic Approach to Model Transformations. In: *EATIS 2008: Euro American Conference on Telematics and Information Systems*, pp. 1–8. ACM, New York (2008)
23. Sendall, S., Kozaczynski, W.: Model Transformation: The Heart and Soul of Model-Driven Software Development. *IEEE Software* 20(5), 42–45 (2003)